From Molecular Dynamics to Oceanography - Ookami Graduate Students Porting and Tuning Science Codes for A64FX

Kedarsh Kaushik*
kedarsh.kaushik@stonybrook.edu
Stony Brook University
Stony Brook, New York, USA

Yuzhang Wang*
yuzhang.wang@stonybrook.edu
Stony Brook University
Stony Brook, New York, USA

Youwei Ma*
youwei.ma@stonybrook.edu
Stony Brook University
Stony Brook, New York, USA

David Carlson
david.carlson@stonybrook.edu
Stony Brook University
Stony Brook, New York, USA

Tony Curtis
anthony.curtis@stonybrook.edu
Stony Brook University
Stony Brook, New York, USA

Robert Harrison
robert.harrison@stonybrook.edu
Stony Brook University
Stony Brook, New York, USA

Eva Siegmann
eva.siegmann@stonybrook.edu
Stony Brook University
Stony Brook, New York, USA

ABSTRACT

To build a cohort of computationally educated young researchers and to deepen community expertise, the Ookami project is actively integrating graduate students into the project team. They learn how to get started on a novel architecture and what to take care of when porting applications. This experience enables them to do production research on A64FX as well as grow their general knowledge in HPC, benefiting not just them but also the codes they are working on. Embedding them within the project team greatly facilitates the transfer of skills, builds a common vocabulary and culture, provides experience in large project execution, and builds confidence and a can-do attitude. In this paper, we report the student's preliminary efforts in porting several science applications to Fujitsu A64FX. The Arm-based Fujitsu A64FX processor developed by Fujitsu and RIKEN is used in Fugaku, which until June 2022 has been the fastest machine worldwide for two years. Its main features of SVE, HBM, and being power efficient makes it unique in the world of HPC. The studied applications (SIESTA, MOM6, Amber, ROMS) are from various disciplines and give a good overview of the A64FX porting process.

CCS CONCEPTS

• **Applied computing** → **Education**;

KEYWORDS

A64FX, high performance computing, porting, Amber, MOM6, ROMS, SIESTA

ACM Reference Format:
Kedarsh Kaushik, Yuzhang Wang, Youwei Ma, David Carlson, Tony Curtis, Robert Harrison, and Eva Siegmann. 2023. From Molecular Dynamics to Oceanography - Ookami Graduate Students Porting and Tuning Science Codes for A64FX. In *Practice and Experience in Advanced Research Computing (PEARC '23), July 23–27, 2023, Portland, OR, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3569951.3593608

*These authors contributed equally to this work.

PEARC '23, July 23–27, 2023, Portland, OR, USA

ACM ISBN 978-1-4503-9985-2/23/07.
https://doi.org/10.1145/3569951.3593608

1 INTRODUCTION - THE OOKAMI TESTBED

In this paper, we investigate the porting and tuning efforts of four different applications for the Fujitsu A64FX processor.

The Ookami testbed [1–3], located at Stony Brook University, is supported by the National Science Foundation (NSF) under grant OAC 1927880. It gives researchers access to 176 Fujitsu A64FX processors. This Arm-based processor is developed by Fujitsu and RIKEN and a variant is used in Fugaku [4]. From June 2020 until June 2022 Fugaku was number one in the Top500 ranking. It also ranked number one in all four other major benchmarks (Green50, HPL-AI, HPCG, Graph500), indicating that the chip is well suited for a wide set of applications, while simultaneously being power efficient.

Ookami has been open to researchers worldwide since January 2021 as a testbed. As of October 2022, it has also been integrated into ACCESS as a resource provider. Experience has shown that most applications compile and run out of the box on this architecture. Getting decent performance, however, can require some effort. This includes testing the available compilers. Their capabilities for vectorizing codes differ a lot and depend on the characteristics of the code.

The Fujitsu A64FX-FX700 processor has a clock speed of 1.8GHz. It is based on Arm v8.2 and has support for SVE (Scalable Vector Extension) with a vector length of up to 512 bits. The processor

consists of 4 NUMA regions, or Core Management Groups (CMG), each having 12 cores, giving a total of 48 cores per node.

By focusing on crucial architectural details, the Arm-based processor with ultrahigh-bandwidth memory promises to retain familiar and successful programming models, while achieving very high performance for a wide range of applications. It supports 64/32/16-bit floating-point representations and fast partial dot-product of 8-bit integers to 32-bit results and hence enables both HPC and big data. On Ookami, 176 nodes are available, facilitated by an HDR100 fat tree interconnect with 200GB switches.

The four different compilers available for the A64FX architecture include Fujitsu (both traditional and clang mode for C and C++), GCC (or GNU), Arm, and Cray (or CPE). The specific vectorization flags for all those compilers are listed in Table 1.

2 EDUCATIONAL INITIATIVES

Since the start of the Ookami project, it was clear that there were multiple barriers to new users making effective use of the system, and, moreover, that many of these concerns were not specific to Ookami. The Ookami team has a multimodal approach [5] to support researchers: (i) an active slack channel, that provides nearly instant help and feedback; (ii) a traditional ticketing system, mainly for installation and project requests; (iii) virtual office hours, lasting for two hours twice a week, for on the spot feedback and support on various topics such as compilation, choosing the right environment, debugging, and profiling; (iv) regular webinars and other virtual training activities; and (v) extensive online documentation and user-oriented FAQ.

Since the beginning of the project, ten graduate students have been employed and integrated into the team. The primary high-level aims were to increase the number of science codes that can make effective use of this technology and to broaden the number of disciplines utilizing the system. However, by partnering with application teams to develop successful ACCESS applications and accelerating the adoption of the system for science, the Ookami team is transferring knowledge into the research science community; translating success on Ookami into performance and productivity advances on other NSF resources; and most importantly, providing substantial educational and research opportunities for the supported students. The students regularly present their progress in the weekly project meetings, as well as in the annual NSF review meeting. They actively join the office hours and already support other new users with their expertise. The Ookami project is also actively encouraging the students to publish their results and findings.

In this paper, the students show their work, development, and their efforts in porting their applications. All of them started with little HPC experience and no exposure to A64FX. Nevertheless, the results demonstrate substantial progress in porting useful applications to this novel architecture.

3 APPLICATIONS

3.1 Amber

Amber is a set of programs widely used in atomic-level molecular dynamic (MD) simulations, especially biomolecules, including proteins and nucleic acids [9]. Molecular simulations can be used to overcome the limitations of wet-lab experiments. It has become a common practice to combine experiments and simulations for researchers in the field of biochemistry. The latest version is Amber22. The core program of Amber, pmemd, is responsible for running MD simulations on high-performance computing platforms. Pmemd supports parallelization by MPI and acceleration by CUDA [10, 11], allowing computing tasks to be distributed on multiple CPU cores and single or multiple GPUs. Amber has been widely tested on x86 CPUs and NVIDIA GPUs.

3.1.1 Compilation. Amber22 uses CMake to prepare the build file for compilation. We compiled Amber on Ookami using GCC (v12.1.0) and Cray (v22.03). Arm (v22.1) and Fujitsu (v4.7) compilers were also tried, which need some modifications in the source code to be compiled successfully, including editing CMake files to support the compilers and fixing some coding errors that are tolerated by GCC but not other compilers. Compilation and testing were focused on pmemd.MPI, the parallel version of pmemd. For comparison with x86 systems, we compiled Amber22 on a computing cluster (Seawulf) with Intel Haswell CPUs with 28 cores.

3.1.2 Molecular systems for testing. Amber supports two types of solvent (water) models, explicit solvent and implicit solvent. We prepared two test systems for each solvent model. For explicit solvent, we chose Dihydrofolate reductase (DHFR). The system consists of 23,558 atoms, among which 2429 are protein atoms. DHFR was simulated at constant volume and constant energy, using a 4 fs step length with hydrogen mass repartition. For implicit solvent, we chose myoglobin, with 2492 atoms in the system. The simulation was conducted at 300K, using the GB-HCT water model [12].

3.1.3 Performance. We used GCC, Arm, and Fujitsu compilers for explicit water systems. The Cray compiler version failed at runtime and is not included in the results. The three compilers performed similarly when using less than 30 cores. When using more than 30 cores, the Arm compiler performed best. All three compilers are slower than the Intel Haswell CPU with 28 cores. The Arm compiler version failed to run on two or more nodes (Fig 1a). Both GCC and Fujitsu showed similar performance on A64FX. Notably, the peak performance for 2 nodes was observed at 46 cores, instead of the available 48 cores per CPU. This is possibly related to the internal architecture of the Fujitsu CPU. We also found a performance drop when using more than three nodes on Ookami. Similar behaviors were also observed on Intel nodes. This could be explained by the limitation of the parallelization strategy of Amber software.

The scaling performance for the implicit solvent system was generally better than explicit water. The Fujitsu compiler was the fastest on a single node among all four tested compilers. It was also observed that using 46 cores per node was the fastest choice on two nodes (Fig 1b). The Arm compiler failed again when using two and more CPUs. The total number of cores was limited to 240 (one-tenth of the number of atoms in the system) due to the software restriction of Amber. When using 46 cores, Amber consumed 103 W of power on an A64FX node while running the DHFR test system.

Generally, the performance of Amber on Ookami is lower than Intel X86 CPUs, but it also comes with the benefit of low energy consumption. With the recent fast development of GPU computing, the GPU version of Amber (pmemd.CUDA) has gained much popularity. However, the computing precision was reduced to a mixed

Table 1: Optimization flags recommended for A64FX

Compiler	Versions	Flags
Arm	21, 21.1, 22.0, 22.0.2, 22.1	*-Ofast -mcpu=a64fx -armpl*
CPE	20.10, 21.03, 22.3, 21.10, 22.03, 22.10	*-O3 -h vector3*
GCC	7.5.0, 9.4.0, 10.2.2, 11.1.0, 11.2.0, 11.3.3, 12.1.0, 12.2.0	*-Ofast -mcpu=a64fx*
Fujitsu	4.2, 4.5, 4.7, 4.8	*-Kfast -KSVE*
Fujitsu clang	4.2, 4.5, 4.7, 4.8	*-Nclang -Kfast -march=armv8.2-1+sve*

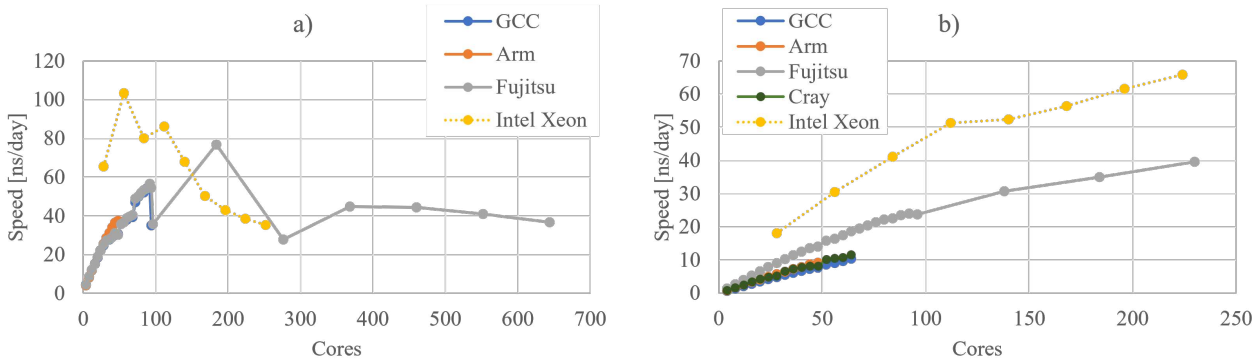

Figure 1: a) Simulation speed of DHFR with explicit water in Amber. b) Simulation speed of myoglobin with implicit water in Amber

single point and double point floating number scheme in the GPU code of Amber. In cases where computing precision is necessary, CPU computing can still be useful.

3.2 SIESTA

SIESTA is an open source *ab-initio* materials modeling Fortran code that applies Density Functional Theory (DFT). DFT can be used to simulate materials that are used in a wide variety of applications including particle detectors, quantum computers, catalysts, drugs, etc. The main computation of DFT involves solving an eigenvalue problem for the electron density self-consistently starting from an initial guess [20, 21].

The widely used diagonalization options are exact diagonalization and OrderN. Exact diagonalization has cubic scaling with system size **O**(N^3) and is implemented using the divide and conquer (D&C) algorithm which divides the matrix into blocks and calculates the eigenvalues of the full matrix from the eigenvalues of the individual blocks. OrderN scales linearly with system size **O**(N). The linear scaling of the latter option can be used to yield accurate results when the localization radius of the orbitals is small compared to the system size and does not work for smaller systems where the range of the orbitals spans the system size. Investigations in this work are restricted to exact diagonalization implemented with the D&C algorithm [20, 22].

3.2.1 Parallel implementation. The running modes supported are serial, MPI, and MPI+OPENMP. Parallelization is controlled by two independent flags `Blocksize` and `ProcessorY`. The former is passed into Scalapack routines to aid diagonalization and the latter is used in speeding up real space 3D integrals computed in the code. In MPI runs, each individual block of the D&C algorithm is assigned a rank and associated memory space. The distribution of the basis into blocks is controlled by the *Blocksize* flag. It can be set by the user, however, the default value is chosen such that the basis elements are as evenly distributed among the MPI ranks as possible. In the hybrid mode, the computation of the individual blocks is distributed amongst the threads which use the shared memory space. The `ProcessorY` flag is used to distribute the mesh points into MPI ranks for numerical integrations performed in the code. The 3D mesh is split into 2+1 dimensions with parallelization over the first 2 dimensions (this is chosen as Y and Z coordinates by default internally). The default value for the `ProcessorY` flag is the integer closest to $\sim \sqrt{N}$. The YZ plane is then approximately distributed into $\texttt{ProcessorY} \times \frac{N}{\texttt{ProcessorY}} = N$ where N is the number of MPI ranks. [22]

3.2.2 Compilation details. Compilation of SIESTA in serial was achieved in all compilers (GCC v11.2, Fujitsu v4.8, Arm v22.0, Cray v22.03) with successful runs. The flags used to compile the code are in Table 1. The optimization levels of the code atom.F reduced to `-O1` for all compilers [22]. Additionally for the Fujitsu compilation, the optimization for mneighb.f also needed to be reduced to `-O1` based on previous installation experience on the Fugaku supercomputer.

Parallel compilation of the application was successfully achieved with the GCC v11.2 + OpenMPI(GCC v11.2), Fujitsu v4.8, and Cray v22.03 + Cray-mvapich2 v2.3.6; however, the compilation with Arm

failed with an internal compiler error. Our compilation experience was similar to what was discussed in [24].

3.2.3 Simulation runs and experience. To test runtimes and scaling, we consider a system of ferroelectric $PbTiO_3/BaTiO_3$ slabs for our results. We choose one small system, with only one formula unit of the crystal with 5 atoms of $\sim$ 4Å size for the test case of serial/low core runs and a slab of $\sim 10nm$ size consisting of 150 atoms for the test case for large parallel runs.

For the serial calculation runs, the wall clock times were 1060 secs, 787.4 secs and 604 secs for the Cray, GCC, and Fujitsu runs respectively. The best runtimes were obtained for the Fujitsu runs.

The parallel runs were successfully performed with the GCC compiled version.

Fujitsu compiler parallel runs were also successful on single compute node jobs with speeds comparable to GCC. However, during multi-node jobs, the calculations terminated at the beginning of the diagonalization step with a SIGBUS error originating from the Infiniband drivers. We eliminated the error by using the `--mca-btl^openib flag` with `mpiexec`. This caused significant slowdowns in the runs as indicated in Fig 2. An update to the Infiniband drivers was able to solve the errors and the Fujitsu runtimes were comparable to the GCC runtimes (see Fig 2).

Fig 2 shows the comparison of 2-node runtimes of single diagonalization steps performed using the GCC(11.2), Fujitsu + FujitsuMPI, Fujitsu + GCC-OpenMPI compilers and Fujitsu + `--mca-btl^openib flag` with `mpiexec`. We see that the Fujitsu compiler suite incl. their compiler and MPI compiler has the fastest runtime for the large system and the GCC compiler has the fastest runtime for the small system.

To test performance across a large number of cores, we report the wall clock times taken for 10 self-consistent iterations which involve 10 diagonalizations in the calculation to get a sense of the average time taken.

We consider parallel runs using the GCC and Fujitsu compilers of the large system with the `Blocksize` and `ProcessorY` flags allowed to be set automatically by the code. If the number of cores (MPI ranks) is c and the number of threads is t, we choose the number or nodes to be **int**((c*t)/48)+1 (as each node has 48 cores). We restrict the number of threads to be 1, 2 or 4 in order to bind the threads to the NUMA regions. The performance of the Fujitsu and GCC compilers show similar runtimes, however, the scaling for both compilers is different.

The Fujitsu compiler shows better performance at a smaller number of cores (c=64, 920 secs) than the GCC compiler (c=200, 975 secs). Further, the GCC runtimes stabilize and remain almost constant for a larger number of cores. The Fujitsu compiler shows a sharper peak at the minima and runtimes increase as the number of cores increases (Fig. 3). For both compilers, threading does not seem to make a large difference in runtimes.

Fig 3 shows the time comparison SIESTA on A64FX and SIESTA compiled using Intel compiler and Intel-MKL math libraries on the x86_64 architecture of the Intel-Haswell cores (available on on-campus Seawulf cluster).

The runtimes on the Intel cores are ~500 secs compared to the ~900 secs minima of the A64FX runs. The runtime remains relatively constant in the range of cores suggesting that the CPU time flattened at a smaller number of cores. The difference in the runtimes could be fixed by compiling the source code with different flags or the difference could be due to diagonalization routines in the scalapack libraries of Fujitsu, GCC, and Intel-MKL.

The calculations performed with the Cray compiler + Cray-mvapich2 failed for all the c/t combinations except for 40/4 (2270.814 secs) and 64/1 (2440.746 secs) where runtimes were $\sim$ 2 times GCC and Fujitsu. The errors were sometimes in the diagonalization (errors in Cholesky decomposition) and sometimes in the initialization. (This issue may be related to similar errors reported for ROMS (sec 3.4) and is beyond the scope of this work).

To summarize, we have successfully ported the SIESTA code to the A64FX architecture on the Ookami computer with the GCC and Fujitsu compilers. However, further code optimization is needed to achieve competitive speeds with Intel cores. Additionally, we have also compiled the PSML version [22] of SIESTA with the GCC compiler. The successful runs with the Fujitsu compiler needed an update to the Infiniband driver, and resulting runtimes were comparable to GCC. Porting to Cray and Arm compilers has not been achieved at the moment. The optimal values for the MPI ranks and threads to be used are strongly dependent on the system and compiler under consideration.

3.3 The Modular Ocean Model version 6 - MOM6

Since the first numerical weather forecast was made in the 1950s using Electronic Numerical Integrator and Computer (ENIAC) [26], the capability of cutting-edge high-performance computation has been a key factor to improve weather forecast and climate prediction. Ocean general circulation models (OGCMs) are significant tools to study and predict the climate. OGCMs are numerical models simulating fluid properties and circulations based on the Navier–Stokes equations on the rotating sphere with thermodynamic terms [27]. According to previous research [28], the advantage of SVE on A64FX may optimize the performance of geophysical fluid simulation. Therefore, our motivation is to optimize OGCMs utilizing the technology of A64FX.

MOM6 is a new-generation OGCM developed at the Geophysical Fluid Dynamics Laboratory (GFDL) of the National Oceanic and Atmospheric Administration (NOAA). MOM6 is widely applied in climate research. For example, MOM6 is the ocean component of GFDL's globally coupled and Earth system models, CM4 and ESM4, respectively. Besides, MOM6 is the new ocean component of the Community Earth System Model (CESM). All these models are used in Version 6 of the Coupled Model Intercomparison Project (CMIP6) supporting the Intergovernmental Panel on Climate Change (IPCC) sixth assessment report (AR6).

We choose the ocean_only/global configuration in the MOM6-example from GFDL's Github repository as our test case.

3.3.1 Compilation. MOM6 is Fortran code and has been compiled successfully using all supported compilers on Ookami including GCC v12.1.0, Arm v22.1, Cray v22.03 and Fujitsu v4.8. The compiling instruction is given on the MOM6-examples wiki webpage. All optimization flags in Table 1 are applied. For GCC v12.1.0, besides the `FFLAGS` in `MOM6-examples/src/mkmf/templates/linux-GNU.mk`, `-fallow-invalid-boz` and `-fallow-argument-mismatch` are added to the makefile. Flag `-fallow-invalid-boz` degrades the

Table 2: Compiler flags for parallel implementation of siesta.(*SF* stands for the serial flags mentioned in Table 1)

Compiler+parallel implementation	Flags
GCC(11.2)+ OpenMPI(Gcc11.2)	*SF + -fopenmp*
Fujitsu(4.8)	*SF + -Kopenmp -SSL2MPI -SSL2BLAMP -SCALAPACK*
Cray(CPE 22.03)+Cray-mvapich2(2.3.6)	*SF + -h omp -eT -G2*

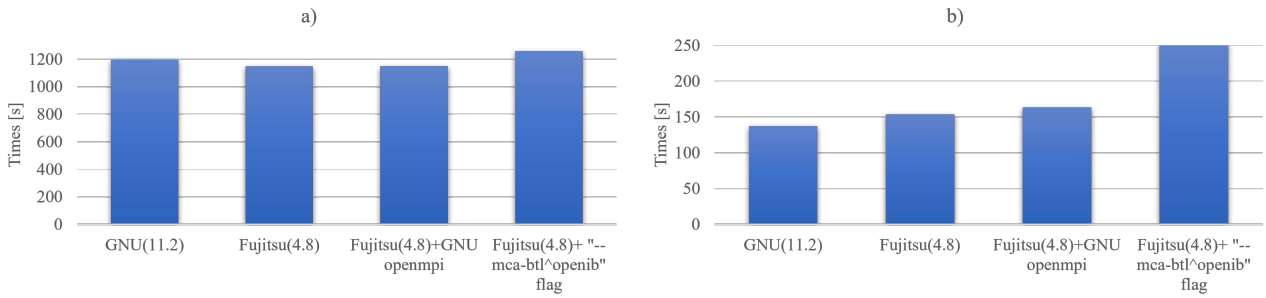

Figure 2: 2-node(4 tasks/node) runtime of the 150 atom system (a) and the 5 atom system(b) for GCC and Fujitsu, Fujitsu+ GCC-OpenMP, Fujitsu+`--mca-btl^openib` flag

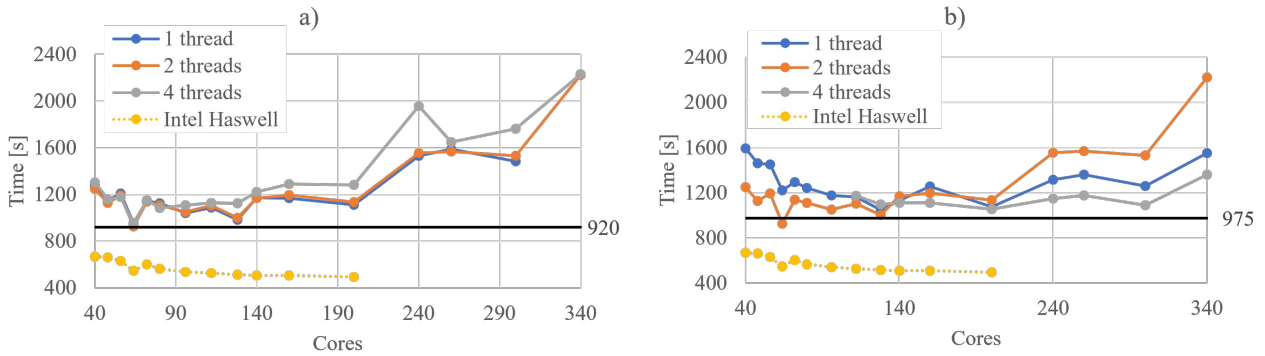

Figure 3: Wall clock times and scaling with number of cores and threads for Fujitsu (a) and GCC (b) compilers. Minimum runtimes are horizontal black lines.

error to a warning and allows binary, octal, and hexadecimal integer constants to appear. The option `-fallow-argument-mismatch` allows the mismatch between the calls and the procedure definition.

For Arm v22.1, we added the rank function to the MPI library using Fortran compiler flag, `-D'rank(X)=size(shape(X))'`. Open MPI v4.1.4 and NetCDF v4.8.1 are both loaded prior to the GCC v12.1.0 and Arm v22.1 compilation. For Cray v22.03, flags are identical to the `MOM6-examples/src/mkmf/templates/ncrc-cray.mk`, plus all optimization flags.

Cray-mvapich2 v2.3.5 and Cray NetCDF v4.7.4.0 are loaded because for MPI library, only Cray-mvapich is available and there is no corresponding Cray openmpi on Ookami. MOM6 supports serial, OpenMP, and MPI computations.

For the MPI + OpenMP computation, the OpenMP flags (`-fopenmp` for Arm and GCC compiler and `-Kopenmp` for Fujitsu compiler) have been added to the makefiles before the compilation. `OPENMP=1` is also required in the `make` commend. As a reference, we also compiled MOM6 with Intel v2022.01 activating `-O3` optimization on Cheyenne, the National Science Foundation (NSF)-supported Cheyenne supercomputer housed at the National Center for Atmospheric Research (NCAR).

3.3.2 Performance Analysis of pure-MPI tasks. We investigate the runtime of a simulation of 30 days realtime compiled with different compilers as a function of the number of cores shown in Figure 4. All the experiments in this section are pure-MPI tasks. For the single-node performance, the Fujitsu compiler outperforms other compilers on Ookami using both 24 cores and 48 cores on a single node. The Fujitsu compiler is two times faster than the GCC compiler. For multi-node performance, the runtime decreases with increasing cores, and the growth rate of runtime with respect to the number of cores decreases and gradually reaches saturation indicating that the performance benefits less from increasing the number of cores. The Fujitsu compiled model is still the fastest for multi-core jobs. The model can be compiled successfully with the Cray compiler but it is only able to run on 1,2 and 5 nodes.

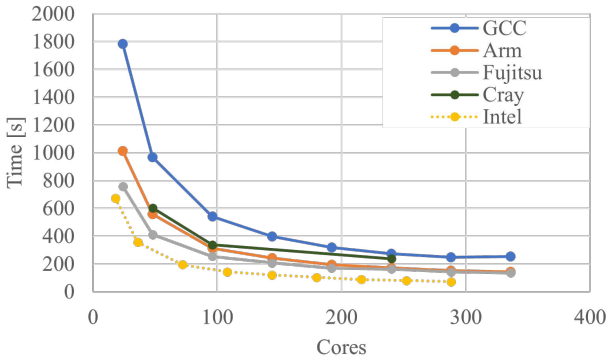

Figure 4: The total runtime of 30-day simulations of MOM6 compiled by GCC, Arm, Cray, Fujitsu on Ookami, and Intel on Cheyenne.

Otherwise, the simulation will be aborted due to a Segmentation fault, which is still under investigation.

Surprisingly, the Intel model running on Cheyenne has a better performance than all the compilers on Ookami. The model throughput using 6 cores (288 processors) is estimated in Table 3. The fewer core hours per simulated year means that with the same simulation time, the model consumes fewer computational resources. Fujitsu-built MOM6 provided a 50-year simulation per wall clock day running on 6 nodes with a power consumption of around 117W per node.

We further allocate fewer MPI tasks per node with a single thread based on NUMA to improve the efficiency of Fujitsu-built MOM6. The cores in idle are evenly distributed on four NUMA regions using `-map-by numa`. We run these experiments on 6 nodes. The total runtime of fully loaded (12 tasks per NUMA) is 143.5 seconds. However, if only 11 tasks are allocated per NUMA, the total runtime decreases to 129.0 seconds (Table 4).

We further use the Fujitsu Profiling Tool to compare more metrics between 12-task-per-NUMA case (case1, hereafter) and 11-task-per-NUMA case (case2, hereafter). The memory throughput peak ratio of case1 (case2) is 38.2% (41.4%). The SVE operation rates for both cases are 85%. `mca_btl_vader_component_progress` is the highest cost application accounting for 29.5% (26.6%) in case1 (case2). The MPI communication of case2 (26.6%) is lower than case1 (36.3%). The above indicates that the MPI communication between two processes via shared memory on the same node is expensive and might have potential room to improve the efficiency of the simulation.

3.3.3 MOM6 MPI+OpenMP hybrid run. To mimic the A64FX layout on Ookami, we use MPI+OpenMP two-level parallelization to minimize interconnect communication for acceleration. The OpenMP-enabled MOM6 is successfully compiled by Arm, GCC, Fujitsu, and Intel compilers. However, it fails to run all compiled versions of MOM6. We and the members of GFDL faculty suspect that it is an initialization bug in the source code. This bug is submitted as #231 issue on the Github page of MOM6-example. Once the bug is fixed, we will test the acceleration using OpenMP threads on A64FX.

3.4 Regional Ocean Modeling System - ROMS

ROMS [6–8] is a free-surface, terrain-following, primitive equations ocean model widely used by the scientific community. It is an open-source Fortran-based code.

The example we are investigating here concerns research on *Pygoscelis* penguins and how their diets reflect the retention of prey around and near their colonies on the west Antarctic Peninsula [17–19]. This region was one of the fastest warming places on the planet [13–16] and how this warming is impacting penguins is still an area of active research. This project will help determine the relationship between penguin diets, local prey retention, and colony presence.

ROMS supports serial, OpenMP, and MPI computations, but no hybrid MPI + OpenMP computations. The domain is partitioned into tiles, where the number of tiles (NtileI and NtileJ) is specified manually by the user. For an MPI job, the product of NtileI and NtileJ must equal the number of MPI processes and for an OpenMP job, the number of threads must be a multiple of the number of tiles. We are using MPI and tried to choose Ntile and NtileJ close to each other, as experience showed the best performance for this configuration.

3.4.1 Compilation. ROMS uses the NetCDF library. Hence, this library was also compiled with all the investigated compilers. All four available compilers with the recommended flags (Table 1) were tested. The Arm v22.0 compiler with OpenMPI v4.1.4 did compile the application without any issues. When running, however, it crashed due to running out of memory. Note that we are restricted to 27GB per node. This is because of the total 32GB available per node, 5GB are reserved for the OS. The Cray v22.03 compiler with Cray-mvapich2 v2.3.6 did compile ROMS. When running a test example the code hung in the initialization step. Further investigation needs to be done to see what causes code initialization to fail. However, this is not in the scope of this paper. The GCC compiler v12.1.0 with OpenMPI v4.1.4 compiled and ran fine. The same is true for the Fujitsu v4.7 compiler in traditional mode.

3.4.2 Performance Analysis. We started by investigating the performance differences between the GCC and the Fujitsu compiled version. As our test setup we are using 32 nodes and 8 tasks per node (see Figure 5). With the GCC compiler, the runtime was 41.13 minutes. With the Fujitsu compiler, it took 30.05 minutes. This substantial performance difference demonstrated the importance of the compiler, as reported by [3, 25].

By default the threads are bound to the core, resulting in the first NUMA region being filled with threads. If the number of threads exceeds 12, the second NUMA region is filled, and so on. The threads can also be distributed between the NUMA regions in a round-robin way. This ensures equal computational load on the NUMA regions. The binding is done by running using the `-map-by numa` option. Using this option the runtime decreased to 26.6 minutes.

The next step was to study the influence of the number of tasks per node. We again used 32 nodes. The number of tasks per node can be increased up to 20 and still fit into the 27GB per node limit. More tasks exceed the memory limitation. NtileI and NtileJ were chosen such that their product equals the total number of MPI

	Core Hour/Simulated Year	Simulated Year/Wall Clock Day
GCC	239.0	28.9
Arm	146.9	47.0
Fujitsu	137.8	50.1
Intel	70.4	98.2

Table 3: Throughput of model compiled by different compilers on 6 nodes

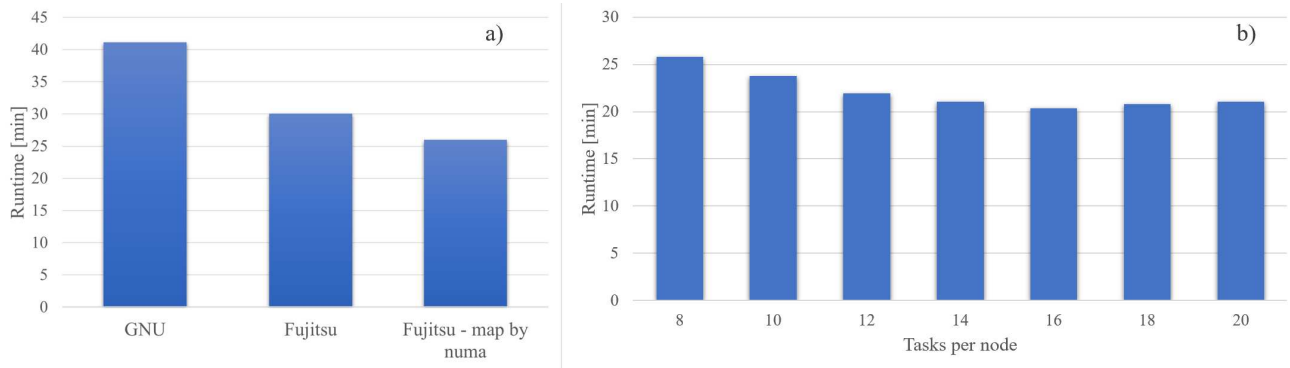

Figure 5: a) Runtime of the GCC compiler as well as of the Fujitsu compiler in different configurations. b) Runtimes on 32 nodes with varying numbers of tasks per node.

Number of tasks per NUMA	Total runtime (seconds)
8	154.8
9	147.4
10	143.5
11	129.0
12	143.5

Table 4: Total runtime of Fujitsu built MOM6 with respect to the number of tasks per NUMA running on 6 nodes.

processes. Figure 5 shows the runtimes for the different setups. The best performance is reached by using 16 tasks per node.

3.4.3 Comparison and Scaling. We compared the performance on A64FX using the Fujitsu compiler with results on Intel Skylake and Intel Haswell nodes. Those nodes are part of an on-campus cluster, to which we have access. The Intel Skylake nodes (Xeon Gold 6148 CPU) with a clockspeed of 2.40GHz consist of 40 cores. The Intel Haswell nodes (Xeon CPU E5-2690 v3) running at 2.60 GHz have 24 cores. The nodes are interconnected via a high-speed InfiniBand FDR network by Mellanox Technologies, allowing transfer speeds of up to 7 Gigabytes of data per second.

On the Intel nodes ROMS as well as NetCDF and OpenMPI were compiled with GCC v12.1.0. We investigated the setup which gave the best performance on A64FX, namely 16x16 tiling running on 32 nodes and using 8 tasks per node. On the Haswell nodes, the calculation finished within 26.4 minutes, whereas on Skylake it took 21.9 minutes.

Do those performance differences sustain when increasing the number of nodes?

For A64FX we could scale up to 128 nodes. For the Intel nodes, the lack of computational resources limited us to 50 nodes. On the Intel

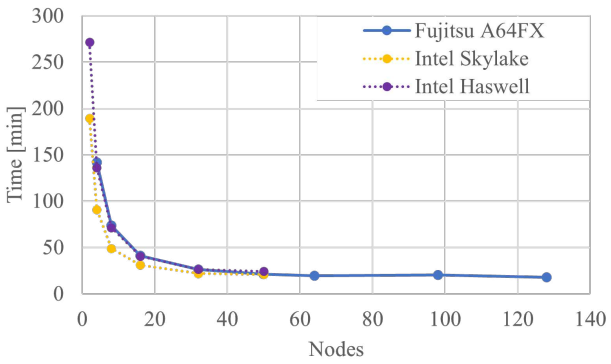

Figure 6: Scaling results on Fujitsu A64FX, Intel Skylake and Intel Haswell

nodes, the setup successfully ran on 2 nodes, whereas on Ookami at least 4 nodes were needed to fulfill the memory requirements of using less than 27GB per node.

Figure 6 shows the runtimes for those runs. Note that each setup was run five times. However, the deviation in runtime was so small that it is not reported in the plot due to visibility (0.5% -2.5%).

On 4 to 32 nodes the Intel Skylake nodes exhibited better performance than the Fujitsu A64FX nodes. At 50 nodes there was performance parity.

We can only speculate that using more than 50 nodes the A64FX nodes might win in terms of performance. Due to the lack of Intel nodes, we could not test this hypothesis at the time of paper submission. However, a performance parity implies a huge power advantage for the Fujitsu nodes.

The Skylake nodes including motherboard, InfiniBand, etc. have a peak power consumption of around 510W. The Fujitsu nodes including all necessary other hardware have a peak consumption of around 180W. We did measure the power consumption for the investigated setups. Measurements were taken once per minute and averaged over the whole runtime. On Fujitsu A64FX ROMS consumed 110.25W and on the Intel Skylake nodes 265.23W. This means that runs on A64FX use 2.4 less power than on Intel Skylake nodes.

4 SUMMARY & OUTLOOK

The paper showcases the work of graduate students from different disciplines as they commence the use of advanced computing technology. From initially being pure HPC users, in the sense that they relied on preinstalled software on a cluster, they are evolving into HPC experts, aware and capable of handling different architectures, with deep knowledge of code design and the cornerstones of performance engineering. They are already transferring this knowledge into their research groups and bringing yet more users into the ACCESS ecosystem. Science codes, which are mainly written by researchers and not by software engineers, will vastly benefit from the users' experience and new know-how.

During the next two years of this project, we will continue to work with new and continuing students. The students will also have the opportunity to attend other NSF-provided cyberinfrastructure training. Many science codes will be ported, tuned, and made available for production on Fujitsu A64FX, with multiple new groups and communities benefiting from national cyberinfrastructure, many for the first time.

ACKNOWLEDGMENTS

The authors would like to thank Stony Brook Research Computing and Cyberinfrastructure, and the Institute for Advanced Computational Science at Stony Brook University for access to the innovative high-performance Ookami computing system, which was made possible by a $5M National Science Foundation grant (#1927880).

REFERENCES

[1] https://www.stonybrook.edu/ookami/
[2] A. Burford, A. Calder, D. Carlson, B. Chapman, F. Coskun, T. Curtis, C. Feldman, R. Harrison, Y. Kang, B. Michalowicz, E. Raut, E. Siegmann, D. Wood, R. Deleon, M. Jones, N. Simakov, J. White, D. Oryspayev; Ookami: Deployment and Initial Experiences; PEARC proceedings 2021
[3] M. Bari, B. Chapman, A. Curtis, R. Harrison, E. Siegmann; A64FX performance: experience on Ookami; 2021 IEEE International Conference on Cluster Computing (CLUSTER), pp. 711-718; 2021
[4] https://www.r-ccs.riken.jp/en/fugaku
[5] E. Siegmann, A. Calder, C. Feldman and R. J. Harrison, "Educating HPC Users in the use of advanced computing technology," 2021 IEEE/ACM Ninth Workshop on Education for High Performance Computing (EduHPC), St. Louis, MO, USA, 2021, pp. 16-23, doi: 10.1109/EduHPC54835.2021.00008.
[6] https://www.myroms.org/
[7] A. Shchepetkin, J. McWilliam; A method for computing horizontal pressure-gradient force in an oceanic model with a nonaligned vertical coordinate, J. Geophys. Res., 108(C3), 3090, 2003
[8] A. Shchepetkin,J. McWilliams; The Regional Ocean Modeling System: A split-explicit, free-surface, topography following coordinates ocean model, Ocean Modelling, 9, 347-404, 2005
[9] Case, Aktulga, Belfon, Ben-Shalom, Berryman, Brozell, Cerutti, Cheatham, III, Cisneros, Cruzeiro, Darden, Duke, Giambasu, Gilson, Gohlke, Goetz, Harris, Izadi, Izmailov, Kasavajhala, Kaymak, King, Kovalenko, Kurtzman, Lee, LeGrand, Li, Lin, Liu, Luchko, Luo, Machado, Man, Manathunga, Merz, Miao, Mikhailovskii, Monard, Nguyen, O'Hearn, Onufriev, Pan, Pantano, Qi, Rahnamoun, Roe, Roitberg, Sagui, Schott-Verdugo, Shajan, Shen, Simmerling, Skrynnikov, Smith, Swails, Walker, Wang, Wang, Wei, Wolf, Wu, Xiong, Xue, York, Zhao, and Kollman (2022), Amber 2022, University of California, San Francisco.
[10] A.W. Goetz, M.J. Williamson, D. Xu, D. Poole, S. Le Grand, and R.C. Walker. (2012) "Routine microsecond molecular dynamics simulations with Amber on GPUs. 1. Generalized Born." J. Chem. Theory Comput. 8, 1542-1555.
[11] R. Salomon-Ferrer, A.W. Goetz, D. Poole; S. Le Grand, and R.C. Walker. (2013) "Routine microsecond molecular dynamics simulations with Amber on GPUs. 2. Explicit solvent Particle Mesh Ewald." J. Chem. Theory Comput. 9, 3878-3888.
[12] V. Tsui; D. A. Case. Theory and applications of the generalized Born solvation model in macromolecular simulations. Biopolymers (Nucl. Acid. Sci.), 2001, 56, 275–291.
[13] Treasure, A. M., Ruzicka, J. J., Pakhomov, E. A., & Ansorge, I. J. (2019). Physical Transport Mechanisms Driving Sub-Antarctic Island Marine Ecosystems. Ecosystems, 22(5), 1069–1087. https://doi.org/10.1007/s10021-018-0326-1
[14] Stammerjohn, S. E., Martinson, D. G., Smith, R. C., & Iannuzzi, R. A. (2008). Sea ice in the western Antarctic Peninsula region: Spatio-temporal variability from ecological and climate change perspectives. Deep Sea Research Part II: Topical Studies in Oceanography, 55(18–19), 2041–2058. https://doi.org/10.1016/j.dsr2.2008.04.02
[15] Ducklow, H. W., Baker, K., Martinson, D. G., Quetin, L. B., Ross, R. M., Smith, R. C., et al. (2007). Marine pelagic ecosystems: the West Antarctic Peninsula. Philosophical Transactions of the Royal Society B: Biological Sciences, 362(1477), 67–94. https://doi.org/10.1098/rstb.2006.1955
[16] Ducklow, H. W., Fraser, W. R., Meredith, M. P., Stammerjohn, S. E., Doney, S. C., Martinson, D. G., et al. (2013). West Antarctic Peninsula: An Ice-Dependent Coastal Marine Ecosystem in Transition. Oceanography, 26(3), 190–203.
[17] Herman, R., Borowicz, A., Lynch, M., Trathan, P., Hart, T., & Lynch, H. (2020). Update on the global abundance and distribution of breeding Gentoo Penguins (Pygoscelis papua). Polar Biology, 43(12), 1947–1956. https://doi.org/10.1007/s00300-020-02759-3
[18] Lynch, H. J., Naveen, R., Trathan, P. N., & Fagan, W. F. (2012). Spatially integrated assessment reveals widespread changes in penguin populations on the Antarctic Peninsula. Ecology, 93(6), 1367–1377. https://doi.org/10.1890/11-1588.1
[19] Cimino, M. A., Lynch, H. J., Saba, V. S., & Oliver, M. J. (2016). Projected asymmetric response of Adélie penguins to Antarctic climate change. Scientific Reports, 6(1). https://doi.org/10.1038/srep28785
[20] José M Soler et al 2002 J. Phys.: Condens. Matter 14 2745
[21] Self-Consistent Equations Including Exchange and Correlation Effects W. Kohn and L. J. Sham Phys. Rev. 140, A1133 – Published 15 November 1965
[22] https://gitlab.com/siesta-project/siesta
[23] https://esl.cecam.org/software/
[24] ISC19_The-First-SVE-Enabled-Arm-Processor-A64FX-and-Building-up-Arm-HPC-Ecosystem.pdf
[25] J. Domke; A64FX – Your Compiler You Must Decide!; arXiv:2107.07157; 2021
[26] Charney, J.G., Fjörtoft, R. and Von Neumann, J. (1950), Numerical Integration of the Barotropic Vorticity Equation. Tellus, 2: 237-254. https://doi.org/10.1111/j.2153-3490.1950.tb00336.x
[27] Chassignet, E. P., LeSommer, J. and Wallcraft, A. J.: Generalcirculation models, Encyclopedia of Ocean Sciences, 3rd edn., edited by: Cochran, KJ, Bokuniewicz, HJ, and Yager, PL, vol. 5, pp. 486-490, 2019
[28] Dongarra, J., 2020: "Report on the Fujitsu Fugaku system" University of Tennessee Knoxville Innovative Computing Laboratory, Tech. Rep. ICLUT-20-06